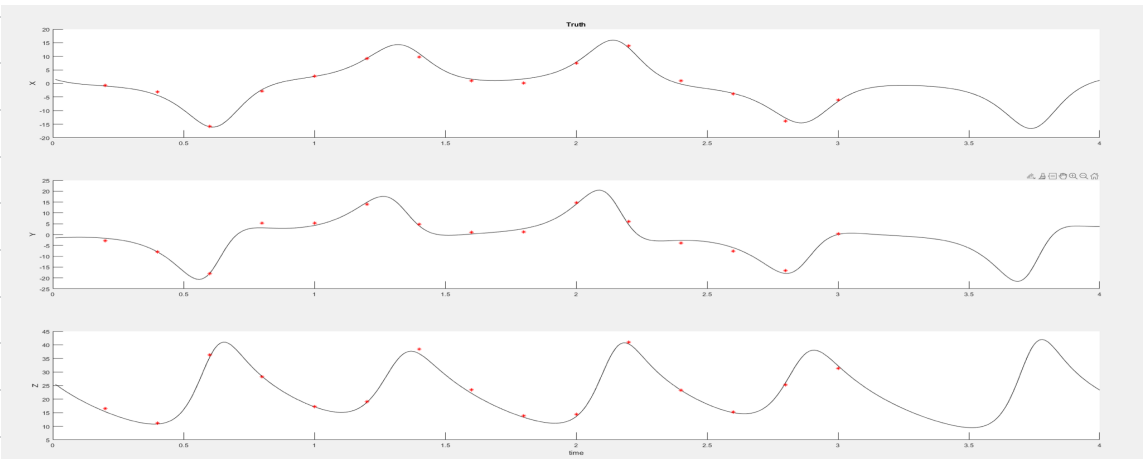
Ndan Stevenson, 109841202
ASEN 6055, Midterm

Start:

1) Implement default Lorenz63_setup_updated and observe:



Understanding:

T = Length of Assimilation window, 3

ft = Length of forecast, 1

dt = Timestep, 0.01

ns = Frequency of obs, 20 (in dts)

truth made w/ $\frac{T+ft}{dt}$ points?

For T+ft time.

Observations are every dt·ns.

Starting after 0 (no meas at t=0)

So total observations are $\frac{T}{dt \cdot ns}$

just the σ for R, P,

$R = \begin{bmatrix} \sigma^2 & & 0 \\ & \ddots & \\ 0 & & \end{bmatrix}$

{ varb = Background variance

varobs = Obs variance

Thus, have our initial state and covariance: x_b, B_{cov}

and observation vector y....

Use this w/ provided forecast model ("modular.m") and TL and ADJ ("tl_adj.m")

2] Do EKF.

Use discrete time pred-update:

Predict [edit]

Predicted state estimate	$\hat{\boldsymbol{x}}_{k\mid k-1} = f(\hat{\boldsymbol{x}}_{k-1\mid k-1}, \boldsymbol{u}_{k-1})$
Predicted covariance estimate	$\boldsymbol{P}_{k\mid k-1} = \boldsymbol{F}_k \boldsymbol{P}_{k-1\mid k-1} \boldsymbol{F}_k^T + \boldsymbol{Q}_{k-1}$

Update [edit]

Innovation or measurement residual	$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_{k\mid k-1})$
Innovation (or residual) covariance	$\boldsymbol{S}_k = \boldsymbol{H}_k \boldsymbol{P}_{k\mid k-1} \boldsymbol{H}_k^T + \boldsymbol{R}_k$
Near-optimal Kalman gain	$\boldsymbol{K}_k = \boldsymbol{P}_{k\mid k-1} \boldsymbol{H}_k^T \boldsymbol{S}_k^{-1}$
Updated state estimate	$\hat{\boldsymbol{x}}_{k\mid k} = \hat{\boldsymbol{x}}_{k\mid k-1} + \boldsymbol{K}_k \tilde{\boldsymbol{y}}_k$
Updated covariance estimate	$\boldsymbol{P}_{k\mid k} = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_{k\mid k-1}$

w/ x_b, B_b initial state and cov.

Given R and Q, $H = I$,

and forward model + jacobian (f, F) given by modular.m + tl_adj.m

Thus, in MATLAB:

```
%% -------- EKF loop ----------
for t = 1:(nSteps-1)
    % ---- FORECAST STEP FOR EVERY DT----

    % Forecasted state
    xf(:,t+1) = modeuler(dt, xa(:,t), A, B, C);

    % Linearize (tangent) about forecasted state:
    [F, ~] = tl_adj(dt, A, B, C, xf(:,t+1));

    % EKF pred
    Pf(:,:,t+1) = F * Pa(:,:,t) * F' + Q;

    % ---- analysis step (if observation available at t+1 and within window T) ----
    if isObsStep(t+1)
        j = obsIndexAtStep(t+1);    % observation index 1..NM
        yo = y(:,j);

        % Innovation
        innov(:,t+1) = yo - H*xf(:,t+1);
        S(:,:,t+1) = H * Pf(:,:,t+1) * H' + R;
        NIS(:,t+1) = innov(:,t+1)' * inv(S(:,:,t+1)) * innov(:,t+1);

        % Kalman gain
        K = Pf(:,:,t+1) * H' / S(:,:,t+1);

        % Analysis state and covariance
        xa(:,t+1) = xf(:,t+1) + K * innov(:,t+1);
        I = eye(nx);
        Pa(:,:,t+1) = (I - K*H) * Pf(:,:,t+1);
    else
        % no observation: carry forecast forward
        xa(:,t+1) = xf(:,t+1);
        Pa(:,:,t+1) = Pf(:,:,t+1);
    end

    % error vs truth
    error(:,t+1) = xa(:,t+1) - xtrue(:,t+1);

    % calculate NEES
    NEES(:,t+1) = error(:,t+1)' * inv(Pa(:,:,t+1)) * error(:,t+1);
end
```

First, analyze default Lorenz.

For comparison metrics, plot x, y, z w/ covs

Then also RMSE, NEES, NIS

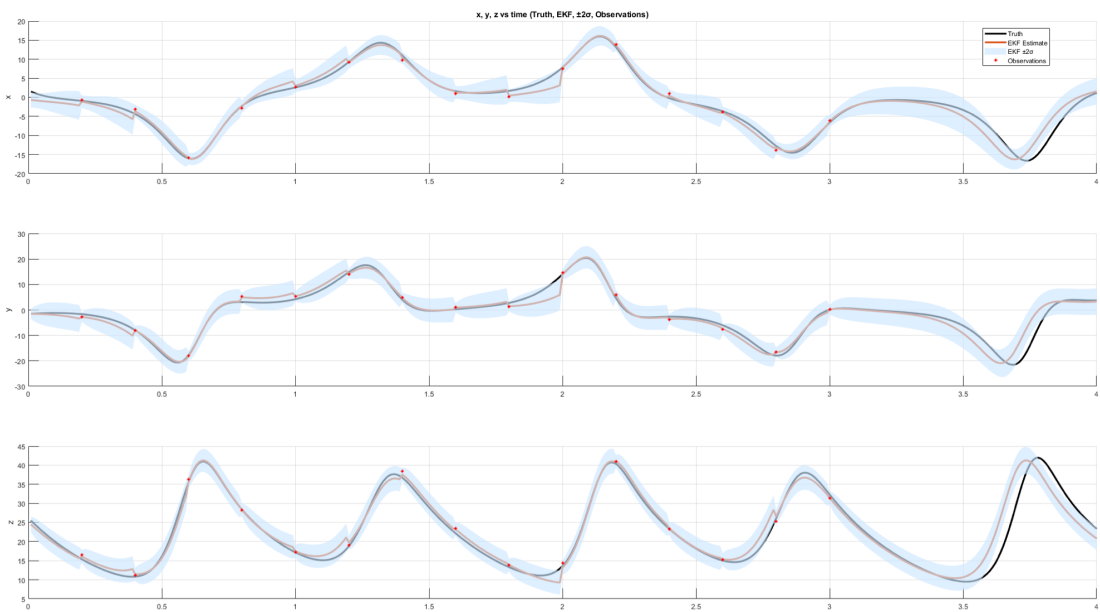
where

$$\text{NEES}: \left(\hat{x}_k - x_k\right)^T P_k^{-1} \left(\hat{x}_k - x_k\right)$$

$$\text{NIS}: \tilde{y}_k^T S_k^{-1} \tilde{y}_k$$

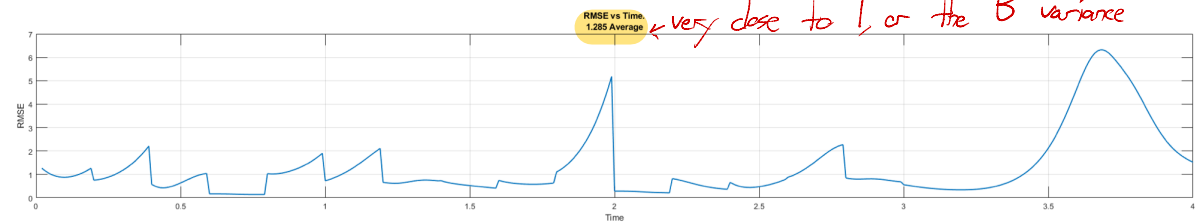
Should follow chi-squared distro. w/ DOF = n.
Compare 95% bounds

Thus, filter output for default:

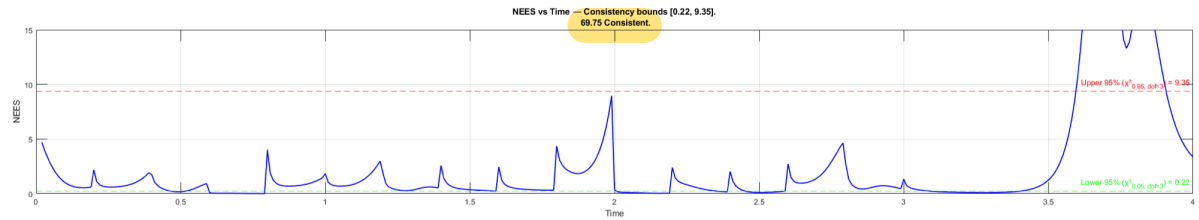


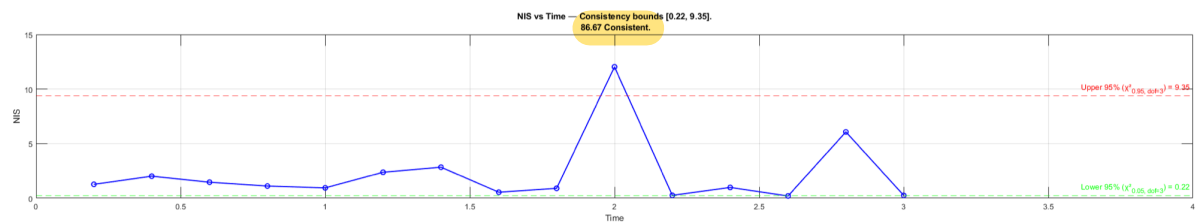
Pretty good tracking. Covs are balanced, and shrink when measurements appear.
At the end (3sec+) we run out of measurements,
thus, causing the filter to diverge from pure predicts.

We can also show RMSE, NEES, NIS performance:



very close to 1, or the B variance





Where ±95% chi2inv bounds are shown.

Thus, we know the goal is 95% consistency in NEES/NIS

NIS is quite well, as we only have a few measurements that give lots of information

NEES struggles from an ending 3-4 seconds w/o any measurements, thus causing the filter to diverge.
(as seen in RMSE)

For all future comparisons (background and accuracy + frequency), we will just be comparing RMSE + NEES

As this represents accuracy + covariance balance

Thus, compare sensitivity of filter performance to errors in background. 10 cases. (assuming can estimate background error)

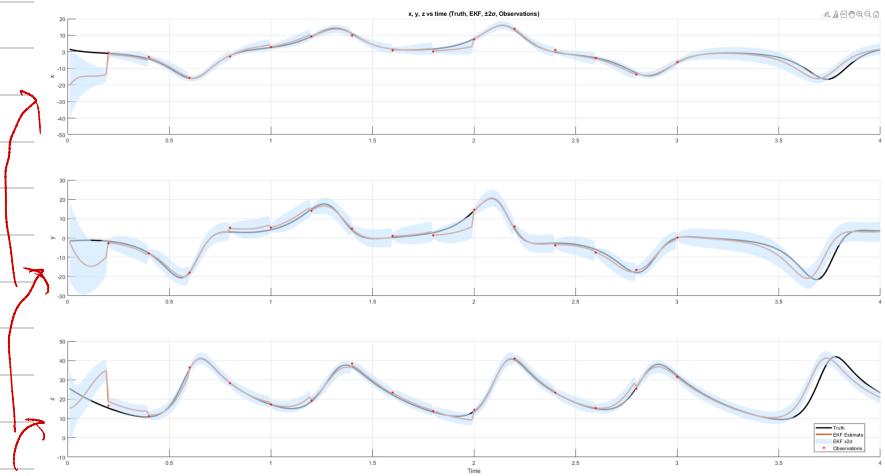
All else is default!

B Variance	RMSE	NEES %	Comments
0.1	1.245	67.25%	Perfect RMSE throughout from great starting error and cov.
1	1.285	69.75%	
2	1.311	69.75%	
3	1.331	69.50%	
5	1.365	69.25%	
7	1.393	69.50%	
10	1.429	69.50%	
20	1.525	67.25%	
50	1.701	66.75%	Very high settling time, high initial cov and error.
100	1.847	66.75%	

Thus, the trend in sensitivity to background variance is obvious.

As the background variance decreases, the RMSE accuracy is better. Higher background variance gives worse accuracy.

This is because, in the given setup file, Bvar is only used to initialize the first estimate + covariance.
Thus, w/ a higher variance, the filter begins in a bad state, as seen w/ Bvar = 100:



Very bad initial state, w/ high starting cov.

What's more important usually is the measurements, thus, now compare sensitivity to frequency and accuracy

RMSE
NEES

outlier

lower is faster

Frequency (in time steps)

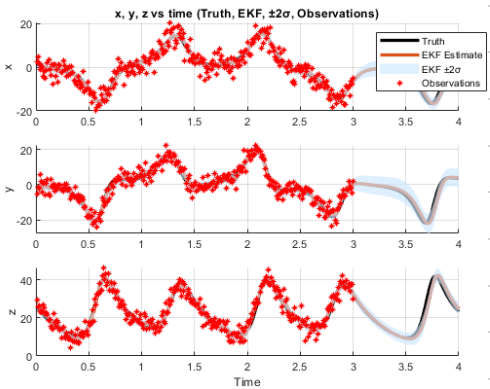
Accuracy (Observation Variance, R)

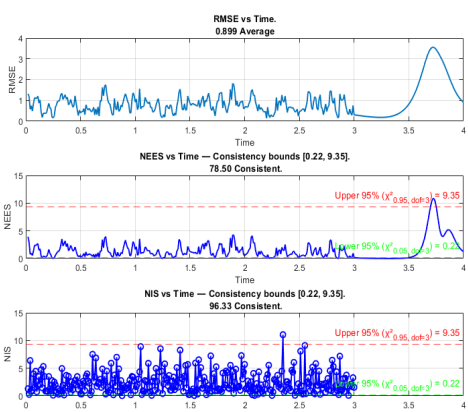
R \ Frequency	1	3	5	10	15	20	30	50	70	100
0.1	0.375 83%	0.21 51%	0.79 41%	0.31 30%	0.362 27%	0.397 28%	1.408 26%	0.767 34%	3.188 32%	2.631 35%
0.5	0.57 84%	0.68 77%	1.85 65%	0.58 62%	0.67 56%	0.85 64%	2.66 41%	1.124 58%	5.3 58%	3.01 48%
1	0.67 81%	1.2 72%	2.6 70%	0.71 70%	0.812 60%	1.28 70%	2.93 47%	1.188 64%	4.77 53%	3.89 52%
2.5	0.94 78%	2.68 73%	2.81 74%	0.81 75%	0.91 66%	2.29 79%	3.05 65%	1.28 77%	7.5 31%	6.3 37%
5	1.11 75%	2.88 77%	2.8 79%	0.89 75%	0.76 61%	3.14 80%	3.3 75%	1.8 75%	9.2 18%	11.4 13%
7.5	1.05 76%	2.9 82%	2.85 72%	0.95 71%	0.91 72%	3.41 70%	3.55 71%	3.84 62%	9.3 17%	8.3 37%
10	0.89 78.5%	2.96 81.5%	2.87 71.2%	0.98 71.2%	0.91 65%	3.5 69.7%	3.76 69.7%	4.28 56%	9.06 20.5%	9.47 31.7%
20	1.0 79	3.2 81	2.4 64	2.1 76	3.6 76	4.5 64	4.8 66	4.8 53	8.4 25	7.22 53
40	1.62 72	3.5 75	1.2 63	4.2 51	3.8 77	4.5 51	10.9 9.5	4.2 65	12.3 9.2	7.0 45
80	1.94 74	3.84 73	2.1 57	6.0 39	2.26 75	6.53 49	11.25 9	3.82 70	12.3 9.2	9.2 20

Analysis on all 100 configs...

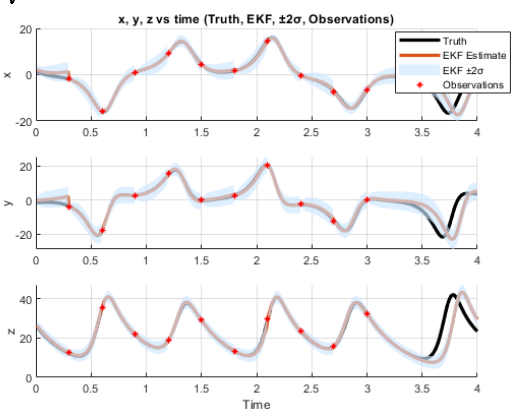
I believe it is better to have frequent, but inaccurate, measurements.
The analysis above does show this.
Take for example Freq 1 w/ R=10 vs Freq 30 w/ R=0.1

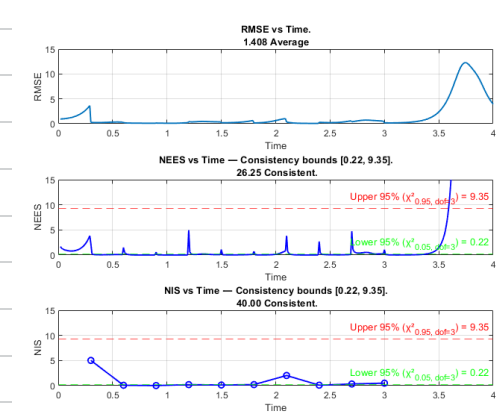
Freq 1, R=10





Freq 30, R=0.1





w/ more measurements, the Kalman filter can estimate the state and cov more accurately, as long as the R is tuned correctly.

NEES, NIS are way up?, and RMSE is better.

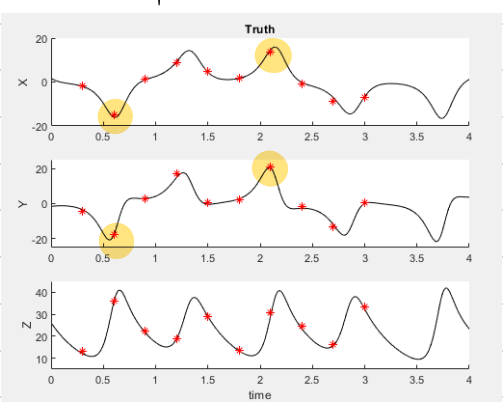
Looking farther at the extremes,

Freq 1, R 80 has 1.94 RMSE, 74% NEES

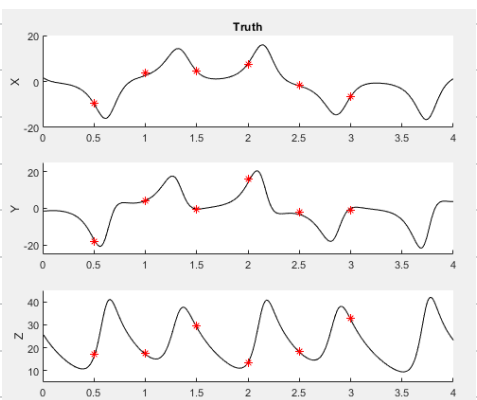
compared to Freq 100, R 0.1 has 2.63 RMSE, 35% NEES.

There are some interesting outliers, such as Freq 30.
Where RMSE is higher compared to Freq 50.
However plots make sense:

Freq 30



Freq 50



Where Freq 30 has unfortunate measurements highlighted at the peaks of the xyz spikes. Thus,

likely causing more error even w/ more meas.

Part 3] EnKF

Now implement EnKF and compare,

First using basic EnKF w/ 50 ensembles.

- Initialize Ensembles by sampling from B_{cov} (background).
- Each ensemble forecasted using provided modeuler.m function, w/ additional errors sampled from $N(0, Q)$ ← provided Q and R
- Then, when we observe, create perturbed obs mat by sampling from $N(0, R)$
- Then update w/ gain from sample cov (P) from ensembles

Thus following given formulation

Suppose we have an ensemble $\{x^{(1)}, \ldots, x^{(i)}, \ldots, x^{(m)}\}$ of our state vector $\boldsymbol{x}$.

Forecast Step:

$$\boldsymbol{x}_t^{f(i)} = \mathcal{M}\left(\boldsymbol{x}_{t-1}^{a(i)}\right) + w_t^{(i)}, \text{ where we draw } w_t^{(i)} \sim N(0, Q_t)$$

Stochastic Update Step (with perturbed observations):

$$y^{f(i)} = \mathcal{H}\left(x^{f(i)}\right) - v_t^{(i)}, \text{ where we draw } v_t^{(i)} \sim N(0, R_t), \text{ to get the state update:}$$

$$x_t^{a(i)} = x_t^{f(i)} + \tilde{K}_t(y_t + v_t^{(i)} - \mathcal{H}\left(x_t^{f(i)}\right))$$

where $\tilde{K} = \tilde{P}^f H^T \left(H\tilde{P}^f H^T + R\right)^{-1}$ with $\tilde{P}^f$ is the ensemble sample covariance matrix

In MATLAB, looks like:

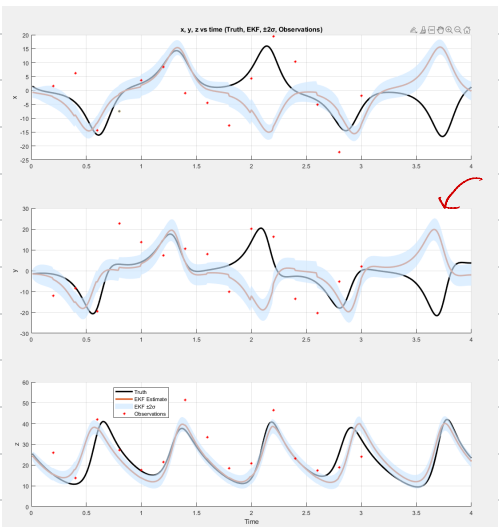
```
%% -------- EnKF Loop ----------
for t = 1:(nSteps-1)
    % ---- Forecast all ensembles ----
    for i = 1:Ne
        En(:,i) = modeuler(dt, En(:,i), A, B, C) + mvnrnd(zeros(nx,1), Q);
    end

    % Inflation mmaybe
    En = mean(En,2) + infl*(En - mean(En,2));

    % ---- UPdate (if obs step) ----
    if isObsStep(t+1)
        j = obsIndexAtStep(t+1);
        y_obs = y(:,j);

        % Ensemble mean and anomalies
        x_mean = mean(En,2);
        Aens = En - x_mean;

        % Sample covariance across all ensembles given the error from mean
        Pf = (Aens * Aens') / (Ne-1);

        % Now gain update - using the sample  cov estimate
        Syy = H * Pf * H' + R;
        K = Pf * H' / Syy;

        % Perturbed observations - or D?
        Y_pert = y_obs + mvnrnd(zeros(size(y_obs)), R, Ne)';

        % Update each ensemble member
        for i = 1:Ne
            innov = Y_pert(:,i) - H * En(:,i);
            En(:,i) = En(:,i) + K * innov;
        end
    end

    % New estimate is mean of the ensemble
    xa_mean(:,t+1) = mean(En,2);

    % just for post (not used in next update) - take the new sample cov of the Ensamp
    Aens_post = En - xa_mean(:,t+1);
    Pa(:,:,t+1) = (Aens_post * Aens_post') / (Ne-1);

    % error vs truth
    error(:,t+1) = xa_mean(:,t+1) - xtrue(:,t+1);
    % calculate NEES
    NEES(:,t+1) = error(:,t+1)' * inv(Pa(:,:,t+1)) * error(:,t+1);
end
```

Where compute a post cov so we can analyze NEES, just like EKF

Now compare to EKF.

Using default lorenz:

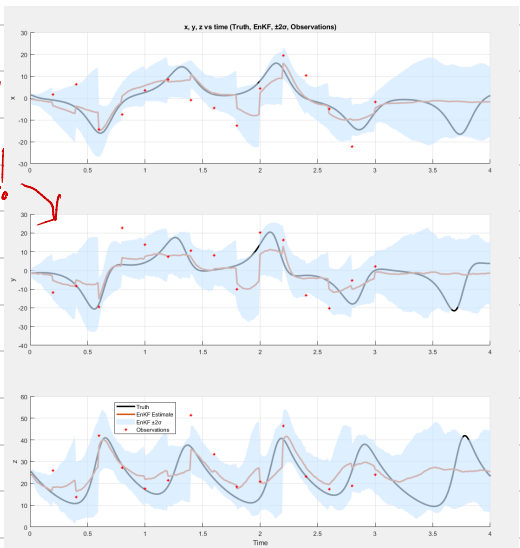
EKF:

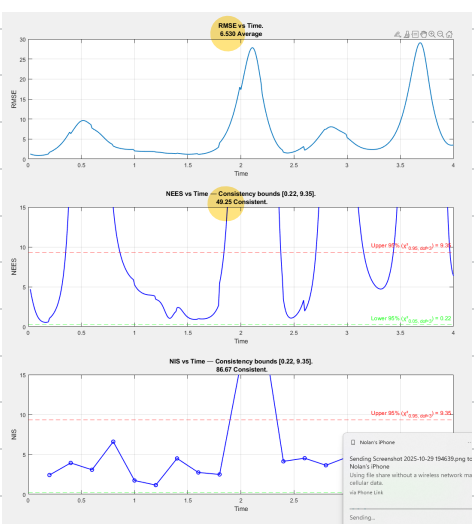


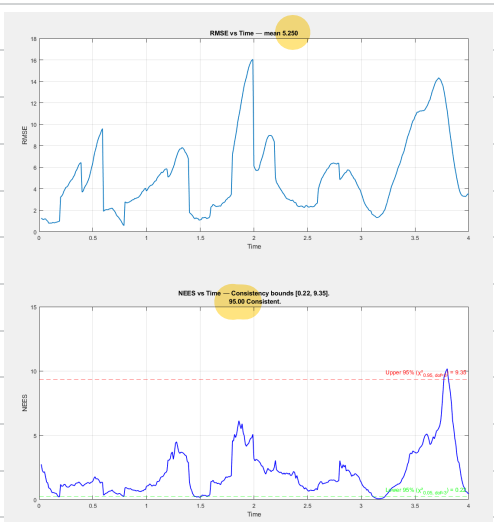
EnKF:

Massive cov difference!

Much better







Across every metric + visual analysis, the EnKF dominates the EKF

- RMSE is down 5.25 compared to 6.53 in the EKF.
- NEES is a perfect 95% consistency compared to just 49%
  - This is obvious visually as the covariances are much better bounded w/ the EnKF. Compared to super overconfident cov estimates by the EKF.
  - The EnKF does a fantastic job of estimating covariance by being better at the non-linearities.

This improvement does make sense due to the ability for the EnKF to properly account for the nonlinear forecast model. Compared to linearizing w/ a jacobian.

Compare some more cases from the above 100 test EKF table

RMSE
NEES

Frequency (in time steps)

Accuracy (Observation Variance, R)

	1	10	30	70
1	EKF: 0.67 61% EnKF 1.9 90%	EKF 0.7 70% EnKF 2.2 71%	EKF: 2.9 47% EnKF 2.6 67%	EKF 4.7 52% EnKF 4.9 84%
5	EKF: 1.1 75% EnKF 2.1 90%	EKF 0.9 75% EnKF 2.6 78%	EKF 3.2 75% EnKF 3.2 86%	EKF 9.2 18% EnKF 5.3 91%
10	EKF 0.9 78% EnKF 2.3 91%	EKF 0.94 71% EnKF 2.9 77%	EKF 3.7 69% EnKF 3.6 90%	EKF 9.1 20% EnKF 5.8 93%
40	EKF 1.6 72% EnKF 2.7 82%	EKF 4.3 51% EnKF 3.9 90%	EKF 10.9 9% EnKF 4.8 94%	EKF 123 9% EnKF 6.7 99%

The comparison b/w the EKF and EnKF is quite interesting. It shows how much better the EnKF is when error is high. All runs w/ R=40 have significantly better NEES consistency, showing the accuracy of the cov estimate as mentioned prior

EKF is lower RMSE for some runs w/ very frequent and low error measurements. But for all infrequent + high error cases, the EnKF is way better on these metrics.

Now, show rank histograms

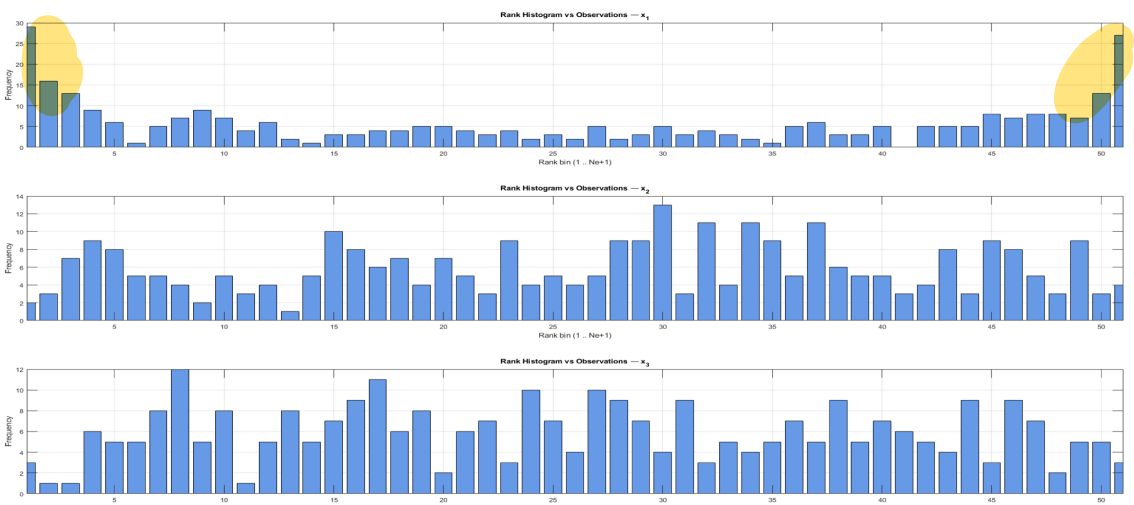
will use freq = 1 to have many observations to rank.

Where we rank meas by ensemble values @ the meas time
- Thus, keep history of ensemble forecasts.

- Ideal is uniform flat line as this means ensembles forecasts are reliable as an observation could be among either of them equally.

And the result for x, y, z is:

with freq = 1



This shows that observations in y and z are quite uniform compared to the forecast ensembles. However, the U-shaped trend in x is concerning. This indicates a bias to the outlier ensembles, which means the forecast isn't spreading out enough. Maybe too small of a Q in x.

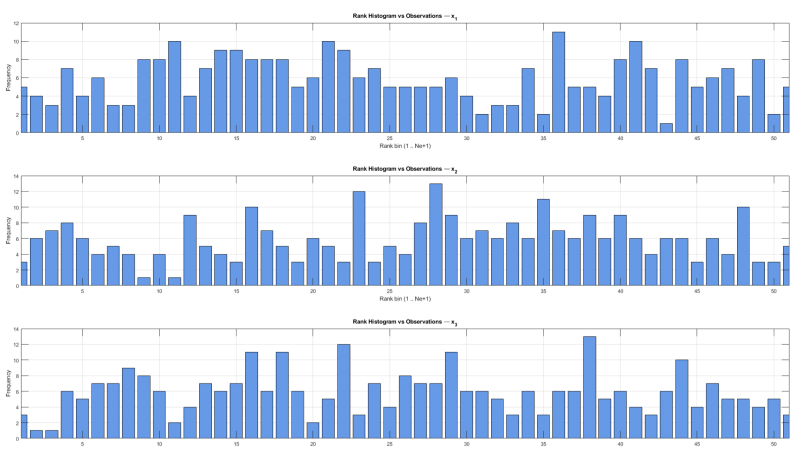
not enough spread

Analyzing further,

If I take the $Q = \begin{bmatrix} 0.15 & \cdots \\ \cdots & 0.9 \end{bmatrix}$

and set $Q(1,1) = Q(2,2) = 0.9$,

We now end up w/ this rank histogram:



Fixed!

Thus proving what I claimed above. Q is too small in x for the given Q

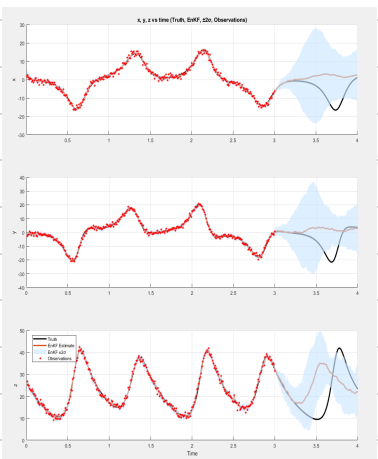
Finally compare different ensemble #s.

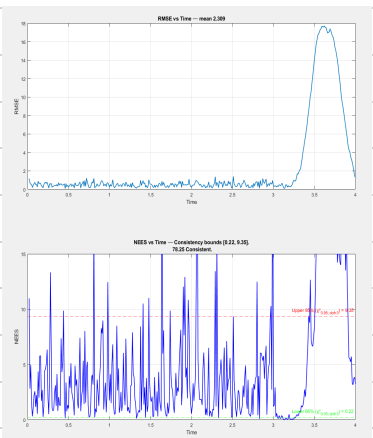
on default, but freq = 1!

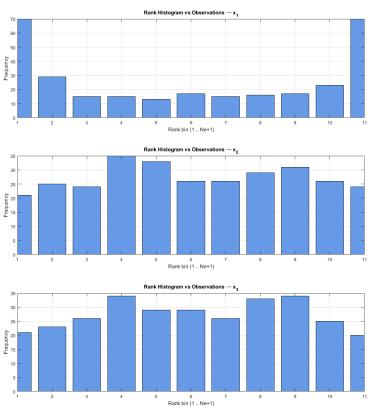
Will compare 10, 50, and 200 ensembles.

In both RMSE + NEES on default params and rank histogram

N=10

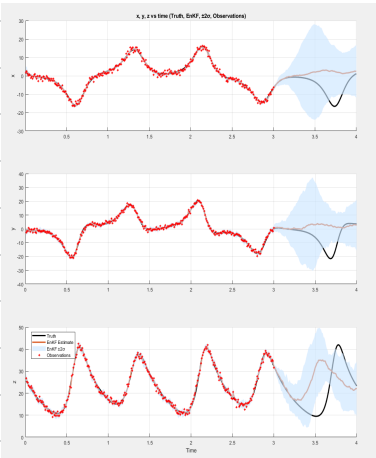


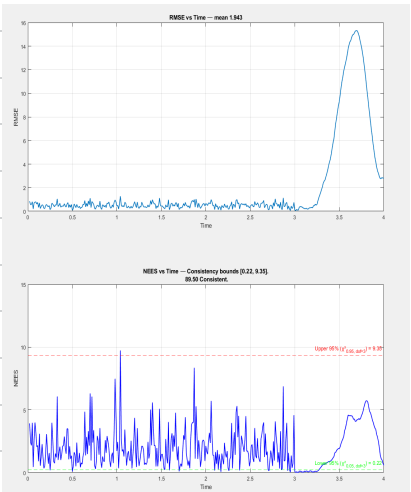


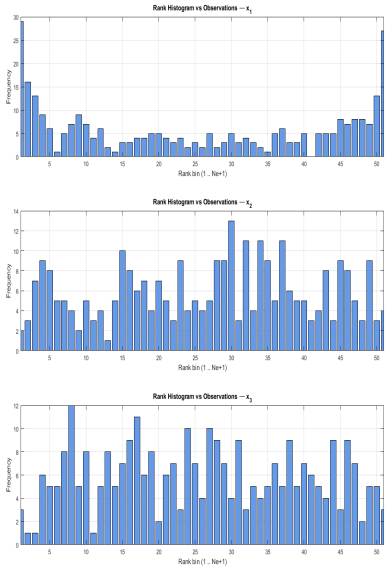


RMSE = 2.3, NEES = 78%

N = 50

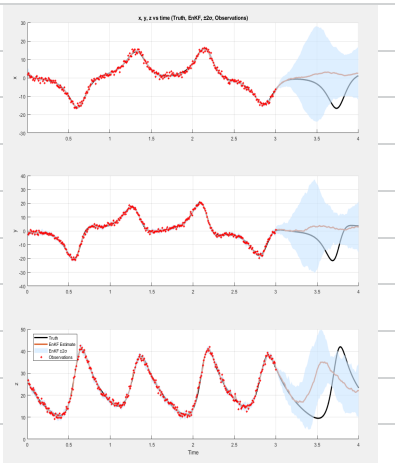


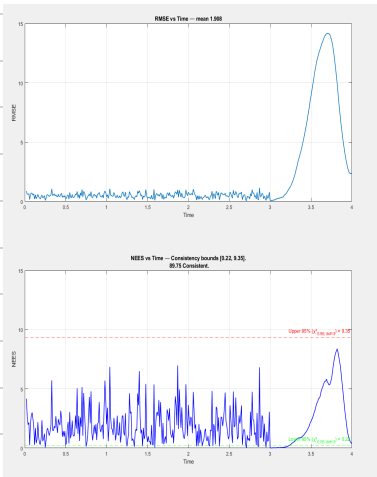


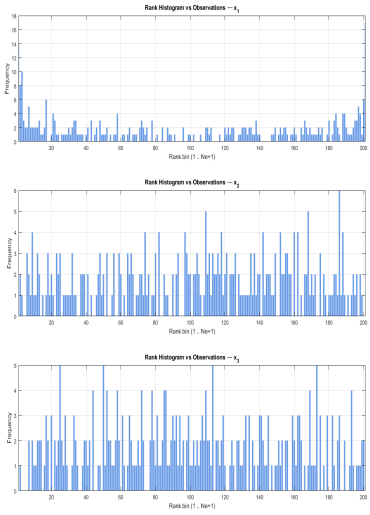


RMSE = 1.94, NEES = 89%

N = 200







RMSE = 1.90, NEES = 90%

Thus, more ensembles does lead to lower RMSE and better NEES. However, no # of ensembles fixes the issue in the rank histogram, as this is primarily a Q tuning problem, as proven above.

Thus, picking # of ensembles does seem to be a tradeoff b/w fidelity and computation.

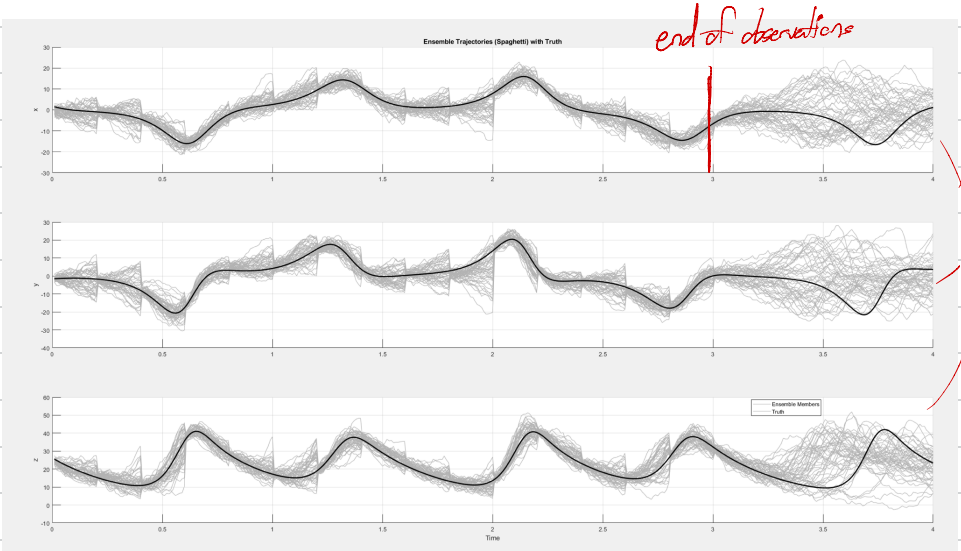
200 had little improvement over 50, at the cost of 4x computation

while RMSE 2.3 → 1.9 (pretty big) when going from 10 to 50 ↑

Finally, I will add the ensemble forecast vs deterministic here.

- Sorry on other plots it shows 1 (deterministic) forecast, I didn't realize what the question meant until now, at the end.

Thus, for N=50, default, we have this total ensemble forecast.



Super cool! Can see how extreme variability becomes.